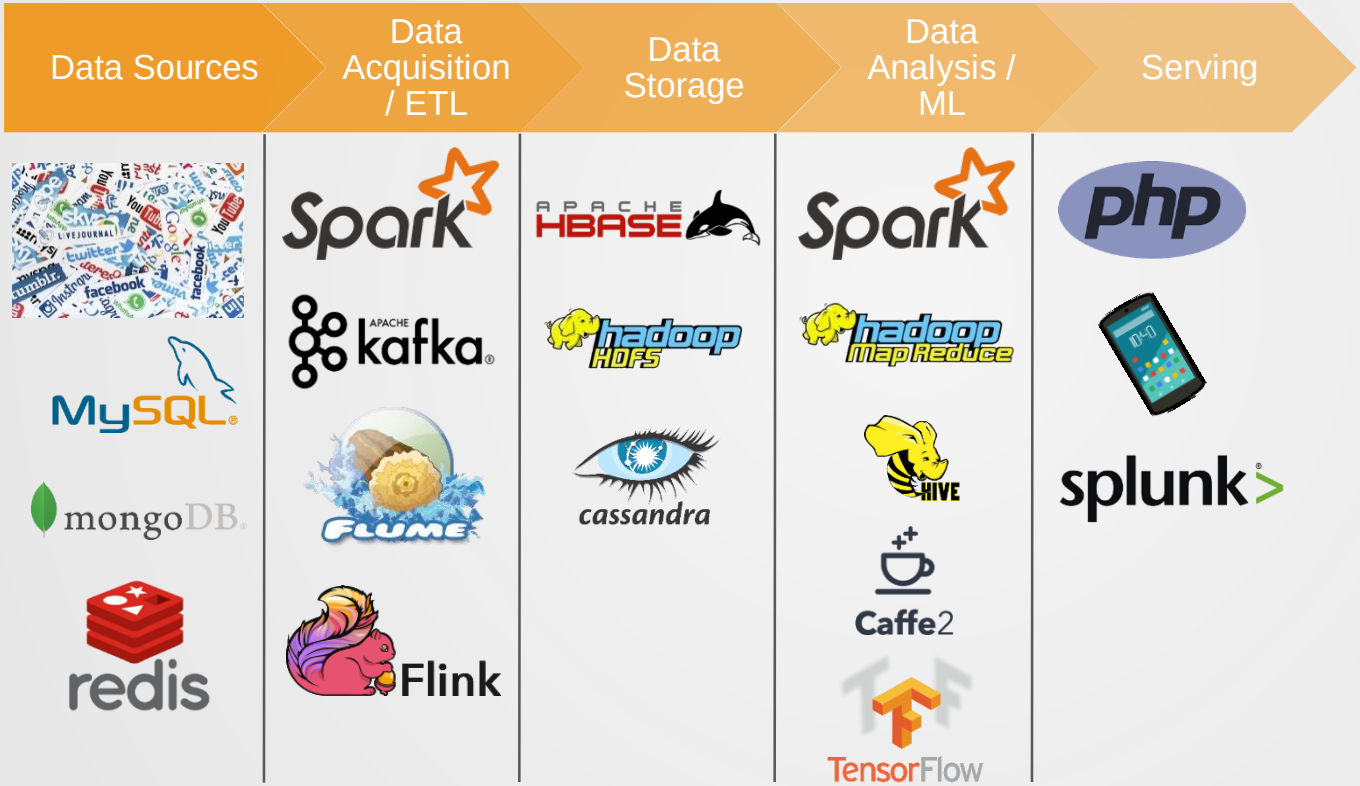
Spark Over RDMA: Accelerate Big Data

SC Asia 2018

Ido Shamay

Apache Spark - Intro

Spark within the Big Data ecosystem



Apache spark 101

Quick facts

- In a nutshell: Spark is a data analysis platform with implicit data parallelism and fault-tolerance
- Initial release: May, 2014
- Originally developed at UC Berkeley's AMPLab
- Donated as open source to the Apache Software Foundation
- Most active Apache open source project

- 50% of production systems are in public clouds

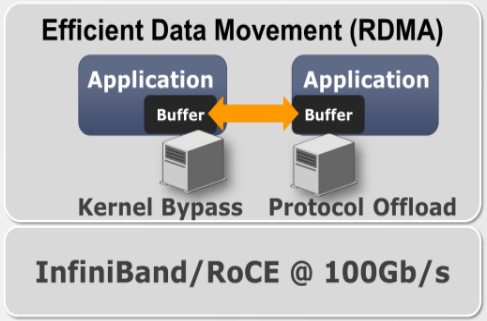
- Notable users:

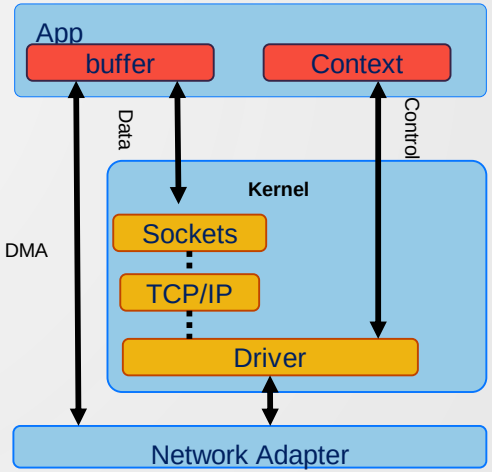
Hardware acceleration in Big Data/Machine Learning platforms

- Hardware acceleration adoption is continuously growing
  - GPU integration is now standard
  - FPGA/ASIC integration is spreading fast
- RDMA is already integrated in mainstream code of popular frameworks:
  - TensorFlow
  - Caffe2
  - CNTK
- Now it's Spark's and Hadoop's turn to catch up

What Is RDMA?

- Stands for “Remote Direct Memory Access”
- Advanced transport protocol (same layer as TCP and UDP)
  - Modern RDMA comes from the Infiniband L4 transport specification.
  - Full hardware implementation of the transport by the HCAs.
- Remote memory READ/WRITE semantics (one sided) in addition to SEND/RECV (2 sided)
  - Uses Kernel bypass / direct user space access
  - Supports Zero-copy
- RoCE: RDMA over Converged Ethernet
  - The Infiniband transport over UDP encapsulation.
  - Available for all Ethernet speeds 10 – 100G
  - Growing cloud support
- Performance
  - Sub-microsecond latency.
  - Better CPU utilization.
  - High Bandwidth.



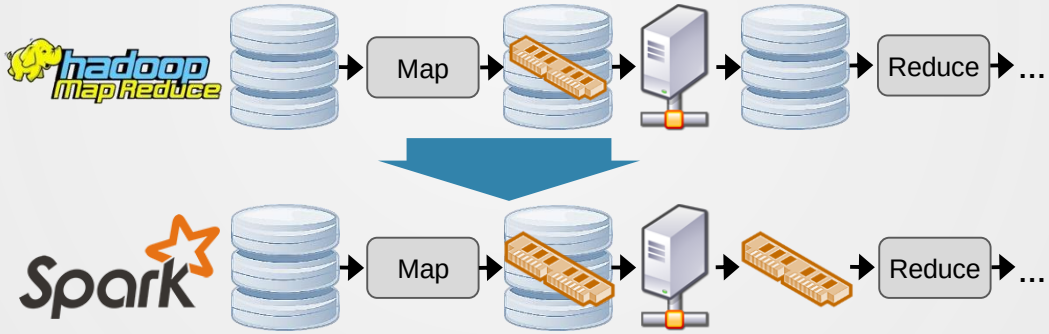


Spark's Shuffle Internals

Under the hood

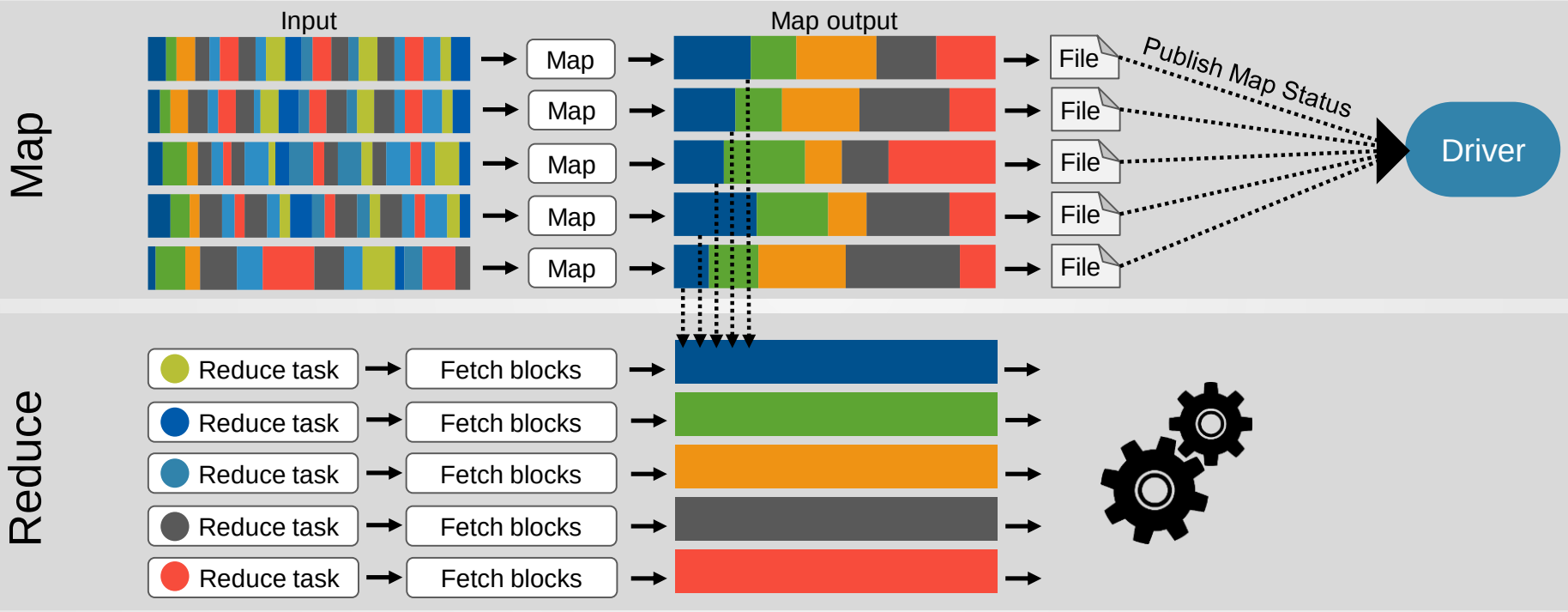
MapReduce vs. Spark

- Spark's in-memory model completely changed how shuffle is done
- In both Spark and MapReduce, map output is saved on the local disk (usually in buffer cache)
- In MapReduce, map output is then copied over the network to the destined reducer's local disk
- In Spark, map output is fetched from the network, on-demand, to the reducer's memory

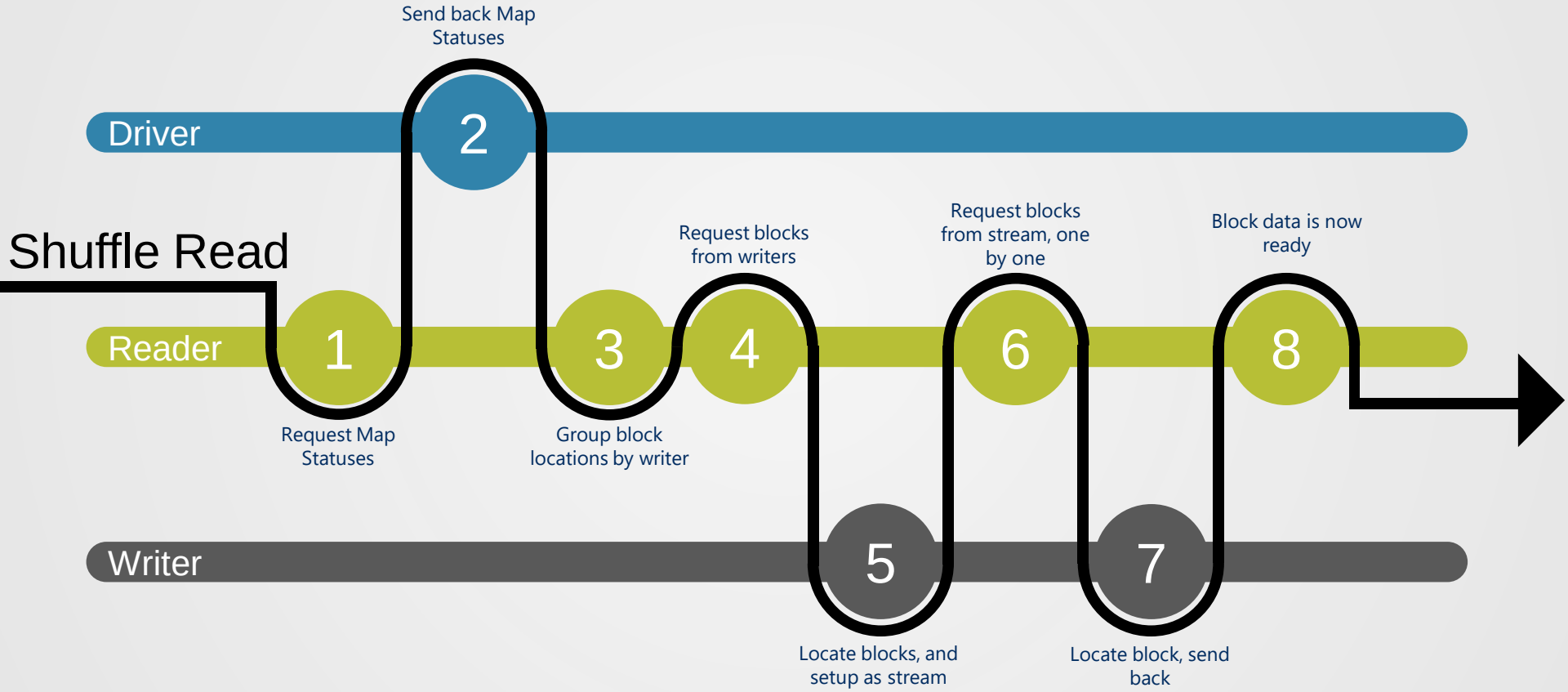


Memory-to-network-to-memory? RDMA is a perfect fit!

Spark's Shuffle Basics



Shuffle Read Protocol



The Cost of Shuffling

- Shuffling is very expensive in terms of CPU, RAM, disk and network IOs
- Spark users try to avoid shuffles as much as they can
- Speedy shuffles can relieve developers of such concerns, and simplify applications

SparkRDMA Shuffle Plugin

Accelerating Shuffle with RDMA

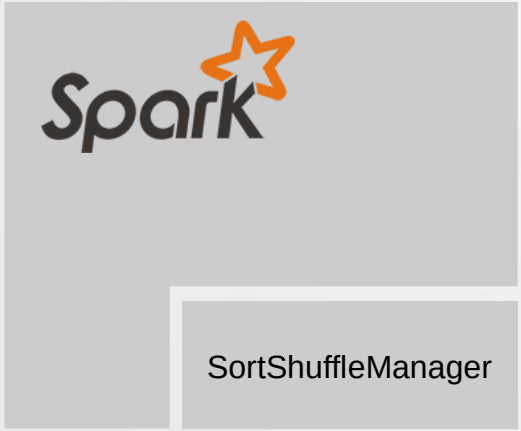
Design Approach

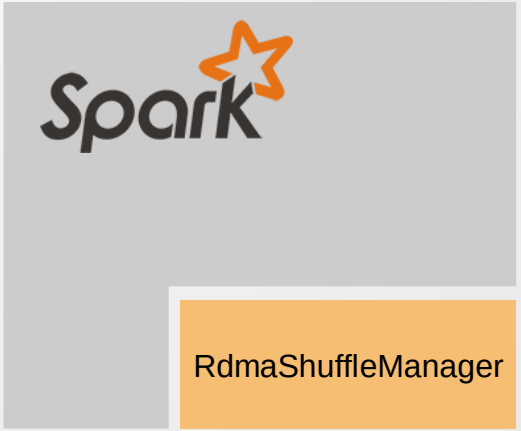
- Entire Shuffle-related communication is done with RDMA
  - RPC messaging for meta-data transfers
  - Block transfers
- SparkRDMA is an independent plugin
  - Implements the ShuffleManager interface
  - No changes to Spark's code – use with any existing Spark installation
- Reuse Spark facilities
  - Maximize reliability
  - Minimize impact on code
- RDMA functionality is provided by "DiSNI"
  - Open-source Java interface to RDMA user libraries
  - https://github.com/zrlio/disni
- No functionality loss of any kind, SparkRDMA supports:
  - Compression
  - Spilling to disk
  - Recovery from failed map or reduce tasks

ShuffleManager Plugin

- Spark allows for external implementations of ShuffleManagers to be plugged in
  - Configurable per-job using: “spark.shuffle.manager”
- Interface allows proprietary implementations of Shuffle Writers and Readers, and essentially defers the entire Shuffle process to the new component

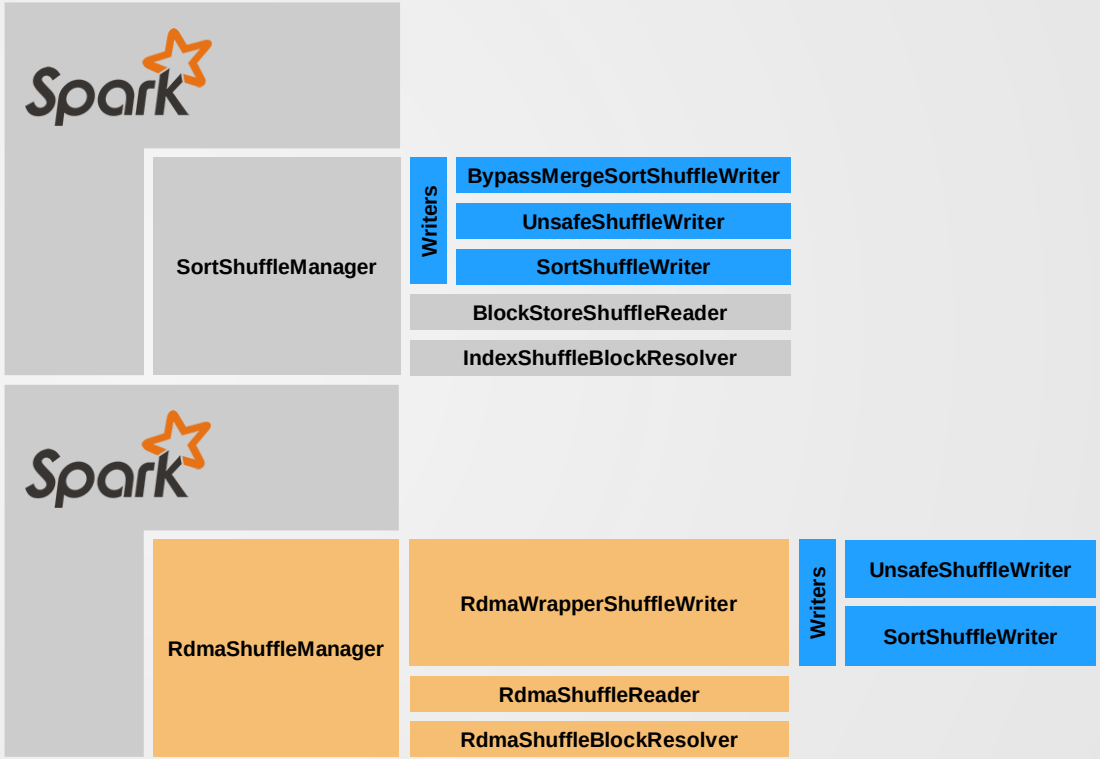
- SparkRDMA utilizes this interface to introduce RDMA in the Shuffle process



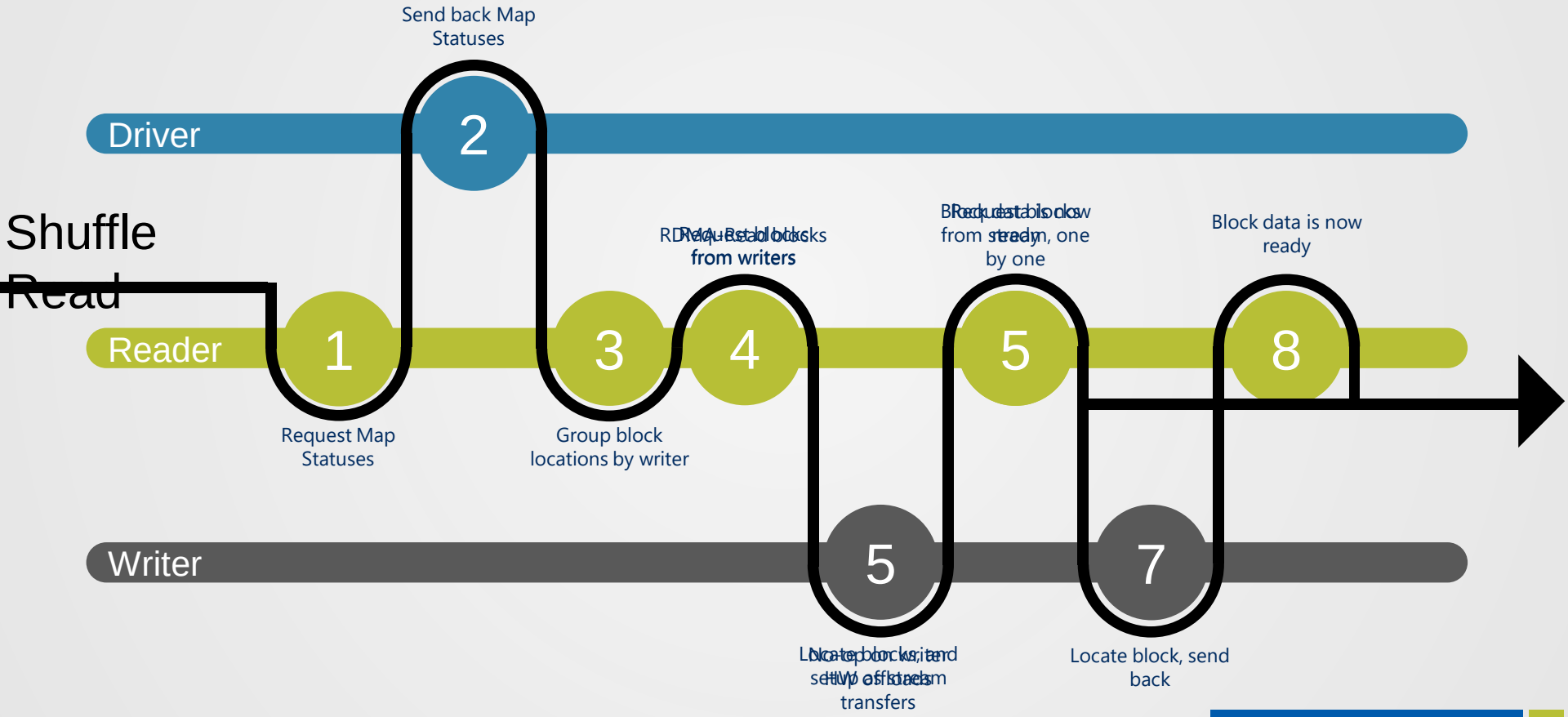


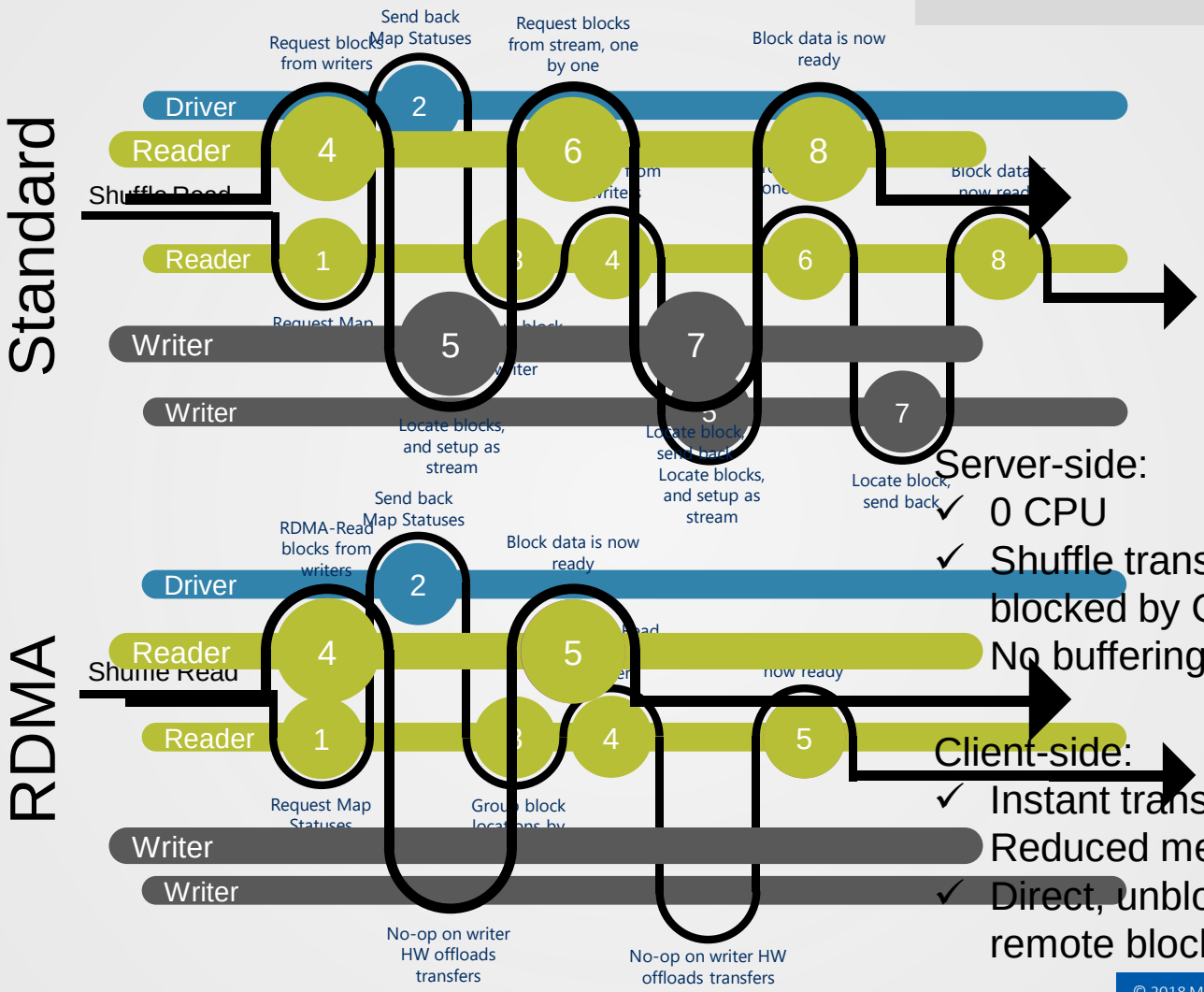
SparkRDMA Components

- SparkRDMA reuses the main Shuffle Writer implementations of mainstream Spark: Unsafe & Sort
- Shuffle data is written and stored identically to the original implementation
- All-new ShuffleReader and ShuffleBlockResolver provide an optimized RDMA transport when blocks are being read over the network



Shuffle Read Protocol – Standard vs. RDMA





Server-side:

- ✓ 0 CPU
- ✓ Shuffle transfers are not blocked by GC in executor
- No buffering

Client-side:

- ✓ Instant transfers
- Reduced messaging
- ✓ Direct, unblocked access to remote blocks

Benefits

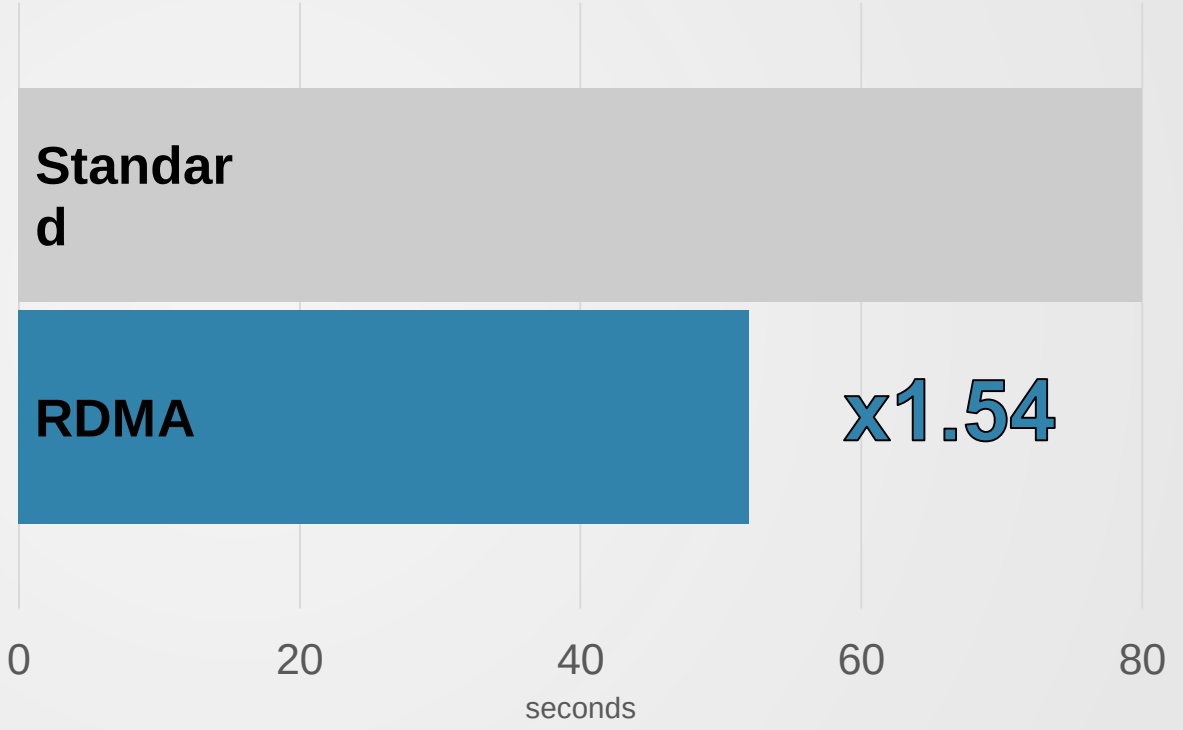
- Substantial improvements in:
  - Block transfer times: latency and total transfer time
  - Memory consumption and management
  - CPU utilization
- Easy to deploy and configure:
  - Supports your current Spark installation
  - Packed into a single JAR file
  - Plugin is enabled through a simple configuration handle
  - Allows finer tuning with a set of configuration handles
- Configuration and deployment are on a per-job basis:
  - Can be deployed incrementally
  - May be limited to Shuffle-intensive jobs

Results

Performance Results: TeraSort

Testbed:

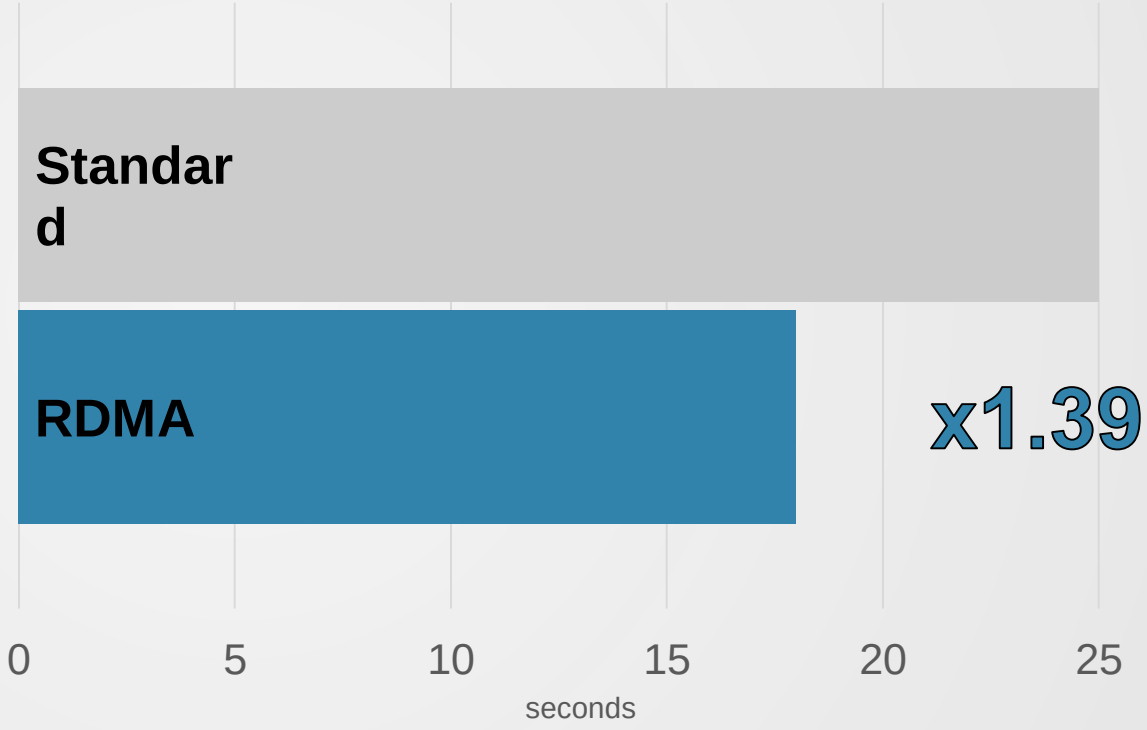
- HiBench TeraSort
  - Workload: 175GB
- HDFS on Hadoop 2.6.0
  - No replication
- Spark 2.2.0
  - 1 Master
  - 16 Workers
  - 28 active Spark cores on each node, 420 total
- Node info:
  - Intel Xeon E5-2697 v3 @ 2.60GHz
  - RoCE 100GbE
  - 256GB RAM
  - HDD is used for Spark local directories and HDFS



Performance Results: GroupBy

Testbed:

- GroupBy
  - 48M keys
  - Each value: 4096 bytes
  - Workload: 183GB
- Spark 2.2.0
  - 1 Master
  - 15 Workers
  - 28 active Spark cores on each node, 420 total
- Node info:
  - Intel Xeon E5-2697 v3 @ 2.60GHz
  - RoCE 100GbE
  - 256GB RAM
  - HDD is used for Spark local directories and HDFS



Coming up next: HDFS+RDMA

HDFS+RDMA

- All-new implementation of RDMA acceleration for HDFS
  - Implements a new DataNode and DFSClient
  - Data transfers are done in zero-copy, with RDMA
- Lower CPU, lower latency, higher throughput
- Efficient memory utilization

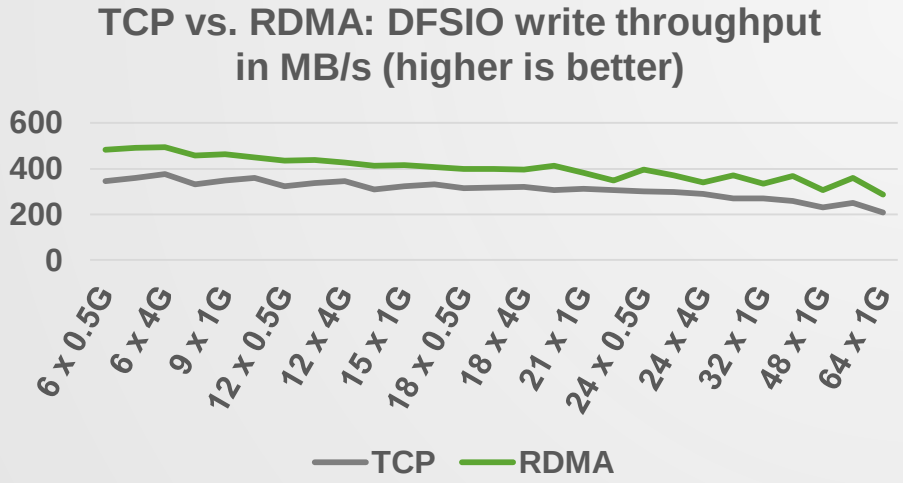
- Initial support:
  - Hadoop: HDFS 2.6
  - Cloudera: CDH 5.10
  - WRITE operations over RDMA
  - READ operations still carried over TCP in this version

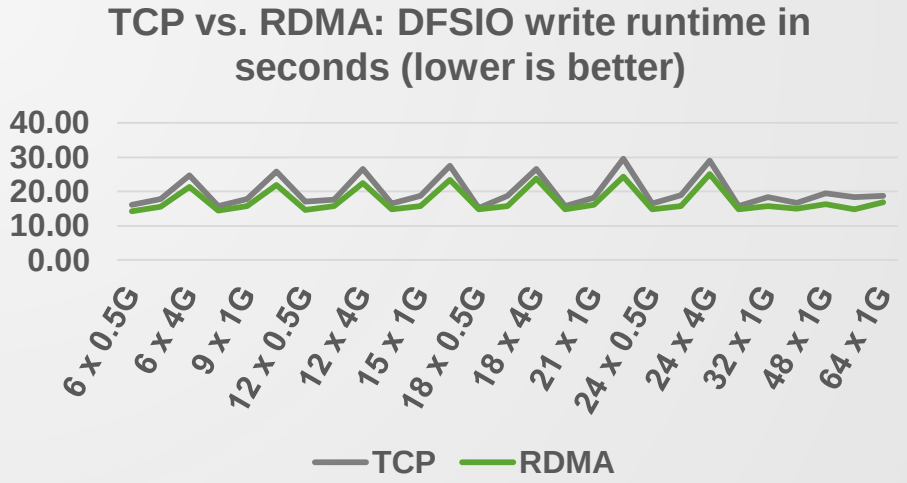
- Future:
  - READ operations with RDMA
  - Erasure coding offloads on HDFS 3.X
  - NVMeF

HDFS+RDMA

- Performance results: DFSIO - TCP vs. RDMA
- CDH 5.10
- 16 x DataNodes, 1 x NameNode
- Single HDD per DataNode
- RoCE 100GbE

- Up to x1.25 speedup in total runtime
- Up to x1.43 in throughput





Roadmap

What's next

Roadmap

- SparkRDMA v1.0 GA is available at https://github.com/Mellanox/SparkRDMA
  - Quick installation guide
  - Wiki pages for advanced settings
- SparkRDMA v2.0 GA – **April 2018**
  - Significant performance improvements
  - All-new robust messaging protocol
  - Highly efficient memory management
- HDFS+RDMA v1.0 GA – Q2 2018
  - WRITE operations with RDMA

Thank You